

Datenbanken und Datensicherheit

Prof. Dr. rer. nat. Peter Gerwinski

18. Dezember 2024

Datenbanken und Datensicherheit

<https://gitlab.cvh-server.de/pgerwinski/dbs>

- 1 Einführung
- 2 Kurzeinführung Unix
- 3 Kurzeinführung TCP/IP
- 4 Relationale Datenbanken
 - 4.1 Einführung in DBMS
 - 4.2 Einführung in SQL
 - 4.3 Normalformen
 - 4.4 Verknüpfungen von Tabellen
 - 4.5 Sichten
 - 4.6 Schlüsselfelder
 - 4.7 Datensicherung
 - 4.8 Aggregate
 - 4.9 Transaktionen
 - 4.10 Indizierung
 - 4.11 Funktionen und Trigger

...

...



Änderungen
vorbehalten

4 Relationale Datenbanken

4.1 Einführung in DBMS

Datenbank-Management-System (DBMS): z. B. PostgreSQL, MariaDB

- Datenbanken „von außen“ verwalten (z. B. auflisten)
- Schnittstelle für Zugriff auf Datenbank
- **in hohem Maße herstellerspezifisch**

```
root@cassini:~# su - postgres
postgres@cassini:~$ psql
psql (15.3 (Debian 15.3-0+deb12u1))
Geben Sie »help« für Hilfe ein.
```

```
postgres=# help
Dies ist psql, die Kommandozeilenschnittstelle für PostgreSQL.
Geben Sie ein:  \copyright für Urheberrechtinformationen
                 \h für Hilfe über SQL-Anweisungen
                 \? für Hilfe über interne Anweisungen
                 \g oder Semikolon, um eine Anfrage auszuführen
                 \q um zu beenden

postgres=#
```

4 Relationale Datenbanken

4.1 Einführung in DBMS

Datenbank-Management-System (DBMS): z. B. PostgreSQL, MariaDB

- Datenbanken „von außen“ verwalten (z. B. auflisten)
- Schnittstelle für Zugriff auf Datenbank
- **in hohem Maße herstellerspezifisch**

```
postgres=# \?  
Allgemein  
  \copyright           PostgreSQL-Urheberrechtsinformationen zeigen  
  \crosstabview [SPALTEN] Anfrage ausführen und Ergebnis als Kreuztabelle  
                        anzeigen  
  \errverbose          letzte Fehlermeldung mit vollen Details anzeigen  
  \g [(OPT)] [DATEI]   SQL-Anweisung ausführen (und Ergebnis in Datei  
                        oder |Pipe schreiben); ...  
  ...                  ...  
  \q                   psql beenden  
  ...                  ...  
  
postgres=#
```

4 Relationale Datenbanken

4.1 Einführung in DBMS

Datenbank-Management-System (DBMS): z. B. PostgreSQL, MariaDB

- Datenbanken „von außen“ verwalten (z. B. auflisten)
- Schnittstelle für Zugriff auf Datenbank
- **in hohem Maße herstellerspezifisch**

```
postgres=# \l
```

Liste der Datenbanken					
Name	Eigentümer	Kodierung	Sortierfolge	Zeichentyp	...
postgres	postgres	UTF8	de_DE.UTF-8	de_DE.UTF-8	
template0	postgres	UTF8	de_DE.UTF-8	de_DE.UTF-8	
template1	postgres	UTF8	de_DE.UTF-8	de_DE.UTF-8	
(3 Zeilen)					...

```
postgres=#
```

4 Relationale Datenbanken

4.2 Einführung in SQL

Datenbank-Abfragesprache: Structured Query Language (SQL)

- Datenbanken „von innen“ verwalten, z. B. anlegen und wieder löschen
- Daten in der Datenbank bearbeiten
- **in hohem Maße standardisiert**

```
postgres=# CREATE DATABASE testdb;
CREATE DATABASE
postgres=# CREATE USER dbsadmin WITH PASSWORD '####';
CREATE ROLE
postgres=# CREATE USER dbs WITH PASSWORD '####';
CREATE ROLE
postgres=# ALTER DATABASE testdb OWNER TO dbsadmin;
ALTER DATABASE
postgres=# \q
postgres@cassini:~$
```

4 Relationale Datenbanken

4.2 Einführung in SQL

Datenbank-Abfragesprache: Structured Query Language (SQL)

- Datenbanken „von innen“ verwalten, z. B. anlegen und wieder löschen
- Daten in der Datenbank bearbeiten
- **in hohem Maße standardisiert**

```
$ psql -h localhost -U dbadmin -W testdb
```

```
Passwort: ####
```

```
psql (15.5 (Debian 15.5-0+deb12u1))
```

```
SSL-Verbindung (Protokoll: TLSv1.3, Verschlüsselungsmethode: TLS_AES_256_G
```

```
Geben Sie »help« für Hilfe ein.
```

```
testdb=> CREATE TABLE tier ( name TEXT, tierart TEXT, id INTEGER );
```

```
CREATE
```

```
testdb=> GRANT SELECT, INSERT, UPDATE, DELETE ON tier TO dbs;
```

```
GRANT
```

```
testdb=> \q
```

4 Relationale Datenbanken

4.2 Einführung in SQL

Datenbank-Abfragesprache: Structured Query Language (SQL)

- Datenbanken „von innen“ verwalten, z. B. anlegen und wieder löschen
- Daten in der Datenbank bearbeiten
- **in hohem Maße standardisiert**

```
$ psql -h localhost -U dbs -W testdb
Passwort: ####
psql (15.5 (Debian 15.5-0+deb12u1))
SSL-Verbindung (Protokoll: TLSv1.3, Verschlüsselungsmethode: TLS_AES_256_G
Geben Sie »help« für Hilfe ein.
```

```
testdb=> DROP TABLE tier;
FEHLER:  Berechtigung nur für Eigentümer der Tabelle tier
testdb=>
```


4 Relationale Datenbanken

4.2 Einführung in SQL

Datenbank-Abfragesprache: Structured Query Language (SQL)

Literatur: z. B. https://de.wikibooks.org/wiki/Einführung_in_SQL

Wichtige SQL-Befehle:

- **CREATE** – Datenbanken, Tabellen usw. anlegen
- **DROP** – Datenbanken, Tabellen usw. löschen
- **SELECT** – Daten abfragen
auswählen mit **WHERE**, sortieren mit **ORDER BY**
- **INSERT INTO ... VALUES** – Daten eingeben
- **UPDATE** – Daten ändern
- **DELETE FROM** – Daten löschen

(Manche DBMS akzeptieren nur Großbuchstaben.)

Felder können auch **NULL** sein (ohne Inhalt).

4 Relationale Datenbanken

4.3 Normalformen

Problem: Schlecht angelegte Datenbanken werden schnell inkonsistent.
Beliebte Fehler:

- Speichern von mehreren Daten in demselben Tabelleneintrag
- Speichern von denselben Daten in verschiedenen Tabelleneinträgen
- implizite Zusammenhänge
- voneinander unabhängige Zusammenhänge in derselben Tabelle

Literatur: z. B. [https://de.wikipedia.org/wiki/Normalisierung_\(Datenbank\)](https://de.wikipedia.org/wiki/Normalisierung_(Datenbank))

4 Relationale Datenbanken

4.3 Normalformen

Problem: Schlecht angelegte Datenbanken werden schnell inkonsistent.
Beliebte Fehler:

- Speichern von mehreren Daten in demselben Tabelleneintrag
→ 1. Normalform
- Speichern von denselben Daten in verschiedenen Tabelleneinträgen
→ 2. Normalform
- implizite Zusammenhänge
→ 3. Normalform und Boyce-Codd-Normalform
- voneinander unabhängige Zusammenhänge in derselben Tabelle
→ 4. und 5. Normalform

Lösung: Normalformen

Literatur: z. B. [https://de.wikipedia.org/wiki/Normalisierung_\(Datenbank\)](https://de.wikipedia.org/wiki/Normalisierung_(Datenbank))

4 Relationale Datenbanken

4.4 Verknüpfungen von Tabellen

Problem: Gut angelegte Datenbanken (→ Normalformen) sind stark aufgesplittet.

Wie kann man sie weiterhin effizient benutzen?

Lösung: Verknüpfungen von Tabellen

SQL-Befehl: JOIN

Literatur: z. B. <https://de.wikipedia.org/wiki/SQL>

4 Relationale Datenbanken

4.4 Verknüpfungen von Tabellen

Was machen wir mit Tabelleneinträgen,
bei denen die **ON**-Bedingung nicht erfüllt ist?

```
SELECT <Feld[er]> FROM <Tabelle1> [INNER] JOIN <Tabelle2>  
    ON <Tabelle1>.<Feld> = <Tabelle2>.<Feld>;
```

→ weglassen: **[INNER] JOIN**

```
SELECT <Feld[er]> FROM <Tabelle1> LEFT JOIN <Tabelle2>  
    ON <Tabelle1>.<Feld> = <Tabelle2>.<Feld>;
```

→ linke Tabelle trotzdem anzeigen (mit **NULL**-Einträgen): **LEFT JOIN**
(analog: **RIGHT JOIN** für rechte Tabelle)

```
SELECT <Feld[er]> FROM <Tabelle1> FULL JOIN <Tabelle2>  
    ON <Tabelle1>.<Feld> = <Tabelle2>.<Feld>;
```

→ beide Tabellen trotzdem anzeigen (mit **NULL**-Einträgen): **FULL JOIN**

4 Relationale Datenbanken

4.5 Sichten

```
SELECT <Feld[er]> FROM <Tabelle1> [INNER] JOIN <Tabelle2>  
    ON <Tabelle1>.<Feld> = <Tabelle2>.<Feld>;
```

→ Wir betrachten beide Tabellen zusammen als eine große Tabelle.

```
CREATE VIEW <Sicht> AS  
    SELECT <Feld[er]> FROM <Tabelle1> JOIN <Tabelle2>  
        ON <Tabelle1>.<Feld> = <Tabelle2>.<Feld>;  
SELECT <Feld[er]> FROM <Sicht> [WHERE ...];
```

→ Wir sprechen das Ergebnis genau wie eine Tabelle an.

→ Es ist möglich, ohne Verlust an Komfort alle Daten in Normalform zu halten.

4 Relationale Datenbanken

4.6 Schlüsselfelder

```
CREATE TABLE tabelle1 (  
    id INTEGER,  
    ...  
);  
CREATE TABLE tabelle2 (  
    ...  
    tabelle1_id INTEGER,  
    ...  
);  
ALTER TABLE tabelle1 ADD CONSTRAINT tabelle1_pkey  
    PRIMARY KEY (id);  
ALTER TABLE tabelle2 ADD CONSTRAINT tabelle2_fkey  
    FOREIGN KEY (tabelle1_id) REFERENCES tabelle1 (id);
```

- Dem DBMS mitteilen, welche Felder für **JOIN** vorgesehen sind.
- Das DBMS kann mit auf Konsistenz achten.

4 Relationale Datenbanken

4.6 Schlüsselfelder

- Dem DBMS mitteilen, welche Felder für **JOIN** vorgesehen sind.
- Das DBMS kann mit auf Konsistenz achten.

Beispiel: Man kann Daten, die noch referenziert werden, nicht löschen.

Alternative: Beim Löschen referenzierter Daten, referenzierende Daten mit löschen.

```
ALTER TABLE tabelle2 ADD CONSTRAINT tabelle2_fkey  
    FOREIGN KEY (tabelle1_id) REFERENCES tabelle1 (id)  
    ON DELETE CASCADE;
```

- Naher Verwandter eines Triggers (kommt gleich)

4 Relationale Datenbanken

4.7 Datensicherung

```
$ pg_dump --clean -h <Rechner> -U <User> -W <Datenbank>
```

- Ausgabe des gesamten Datenbankinhalts als SQL-Quelltext zur Standardausgabe

—> keine Probleme mit sich evtl. ändernden Binärformaten

- Es ist möglich, den Inhalt direkt in einer Pipe weiterzuverarbeiten (z. B. zu komprimieren).
- Zurückspielen: mit `psql`

```
$ psql -h <Rechner> -U <User> -W <Datenbank> \  
    < <Ausgabe von pg_dump>
```

- analog für **MariaDB**: `mariadb-dump`

4 Relationale Datenbanken

4.8 Aggregate

Rechenoperationen auf der ganzen Spalte, z. B. Summenbildung:

```
testdb=> SELECT SUM (id) FROM tier;
```

sum

4143

Rechenoperationen: SUM, AVG, MIN, MAX, COUNT

Gruppierung:

```
testdb=> SELECT tierart, MIN (id) AS "Minimum" FROM tier
        GROUP BY tierart;
```

tierart	Minimum
Spinne	94
Hund	7
Kaninchen	42

Auswahl mit der anzuzeigenden Gruppen mit HAVING (entsprechend WHERE)

4 Relationale Datenbanken

4.9 Transaktionen

- Ziel: Wahrung der Konsistenz
- Methode: zusammengehörige Aktionen zu einer *Transaktion* zusammenfassen
- Beispiel: Überweisung
Betrag von einem Konto subtrahieren
und „gleichzeitig“ zu einem anderen Konto addieren
- Realisierung in PostgreSQL: `BEGIN; ... COMMIT;`
- Realisierung in MariaDB: `START TRANSACTION; ... COMMIT;`
- Abbruch einer Transaktion: `ROLLBACK;` statt `COMMIT;`

4 Relationale Datenbanken

4.10 Indizierung

- Ziel: Performanzgewinn
- Methode: Spalten, in denen häufig gesucht wird, sortieren
- Beispiel: Datenbank mit Kontaktdaten
Suche nach Name, Adresse, Telefonnummer
- Realisierung in PostgreSQL und MariaDB:

```
CREATE INDEX <Indexname> ON <Tabellenname>
(
    <Spaltenname>,
    ...
);
```

- Vorteil: schnellere Suche
(**SELECT**) in indizierten Spalten

Suche nach String:

Zeilen	ohne Index	mit Index
1000	1.809 ms	2.022 ms
1000000	501.546 ms	1.264 ms

4 Relationale Datenbanken

4.10 Indizierung

- Ziel: Performanzgewinn
- Methode: Spalten, in denen häufig gesucht wird, sortieren
- Beispiel: Datenbank mit Kontaktdaten
Suche nach Name, Adresse, Telefonnummer
- Realisierung in PostgreSQL und MariaDB:

```
CREATE INDEX <Indexname> ON <Tabellenname>
(
    <Spaltenname>,
    ...
);
```

- Vorteil: schnellere Suche
(**SELECT**) in indizierten Spalten
- Nachteil: langsames Einfügen/Ändern/Löschen

Suche nach String:

Zeilen	ohne Index	mit Index
1000	1.809 ms	2.022 ms
1000000	501.546 ms	1.264 ms